Introdução ao framework Python flask

Prof. Me. Hélio Esperidião

O que é um framework?

- Framework é um termo que se refere a estratégias e ações que visam solucionar um tipo de problema.
- é uma estrutura que serve de base para a construção de aplicações de finalidade específica cujo desenvolvimento pode ser muito custoso e/ou problemático.
- Com um framework é possível construir softwares a partir de um esqueleto pré-definido, alterando apenas demais particularidades.

Framework

- Um framework é um conjunto de ferramentas prontas para uso que oferece componentes para resolver problemas e superar desafios de desenvolvimento de forma rápida e altamente funcional. Em programação, um framework geralmente é composto por bibliotecas de código, APIs, documentação de referência, um compilador e outras ferramentas de suporte..
- Esses recursos ajudam os desenvolvedores a acelerar o processo de criação de aplicativos, fornecendo uma estrutura sólida e abstrata para lidar com tarefas comuns, permitindo que eles se concentrem mais na lógica de negócios específica do aplicativo..


Springboot
Bootstrap
jQuery
Cordova
Corona
Frameworks para o Desenvolvimento de Softwares
Angular
Flutter
React
Material Design
Ionic

Por que usar um framework?

- Uma das principais vantagens de utilizar um framework é a sua eficiência em termos de tempo.
- Os frameworks de desenvolvimento web permitem que os desenvolvedores economizem energia e tempo ao lidar com diversas funcionalidades.
- Ao fornecer componentes pré-construídos e soluções prontas para uso, os frameworks agilizam o processo de desenvolvimento, permitindo que os programadores se concentrem mais na lógica do aplicativo em vez de ter que reinventar a roda para cada funcionalidade.
- Isso resulta em uma maior produtividade, redução de erros e tempo de desenvolvimento mais curto, o que é extremamente valioso em projetos com prazos apertados.

Principais frameworks web

- React : Javascript > front-end.
- Express JS: Javascript > back-end
- Django: python > full-stack
- Ruby on Rails: Ruby > full-Stack
- Laravel: php > full-Stack
- Spring: java >full-Stack
- Angular:Javascript > front-end.
- Vue JS: Javascript >front-end.
- Asp.Net: multlinguagem, tipicamente c# no backend.

Flask

- O Flask é um framework web em Python, conhecido por sua simplicidade e flexibilidade.
- Foi projetado para ser um microframework, o que significa que possui um núcleo básico, mas pode ser facilmente estendido com ferramentas e bibliotecas adicionais, se necessário.

Crie um diretório

- Crie um diretório para abrigar sua aplicação.
  - C:\apiTeste

Instalar o flask

- pip install flask


```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [versão 10.0.19044.2965]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Helio>cd c:\apiteste

c:\apiTeste>pip install flask
Requirement already satisfied: flask in c:\users\helio\appdata\local\programs\python\python311\lib\site-packages (2.3.2)

Requirement already satisfied: Werkzeug>=2.3.3 in c:\users\helio\appdata\local\programs\python\python311\lib\site-packag
es (from flask) (2.3.6)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\helio\appdata\local\programs\python\python311\lib\site-packages
 (from flask) (3.1.2)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\users\helio\appdata\local\programs\python\python311\lib\site-pa
ckages (from flask) (2.1.2)
Requirement already satisfied: click>=8.1.3 in c:\users\helio\appdata\local\programs\python\python311\lib\site-packages
(from flask) (8.1.3)
Requirement already satisfied: blinker>=1.6.2 in c:\users\helio\appdata\local\programs\python\python311\lib\site-package
s (from flask) (1.6.2)
Requirement already satisfied: colorama in c:\users\helio\appdata\local\programs\python\python311\lib\site-packages (fro
m click>=8.1.3->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\helio\appdata\local\programs\python\python311\lib\site-packag
es (from Jinja2>=3.1.2->flask) (2.1.3)

[notice] A new release of pip available: 22.3.1 -> 23.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip

c:\apiTeste>
```

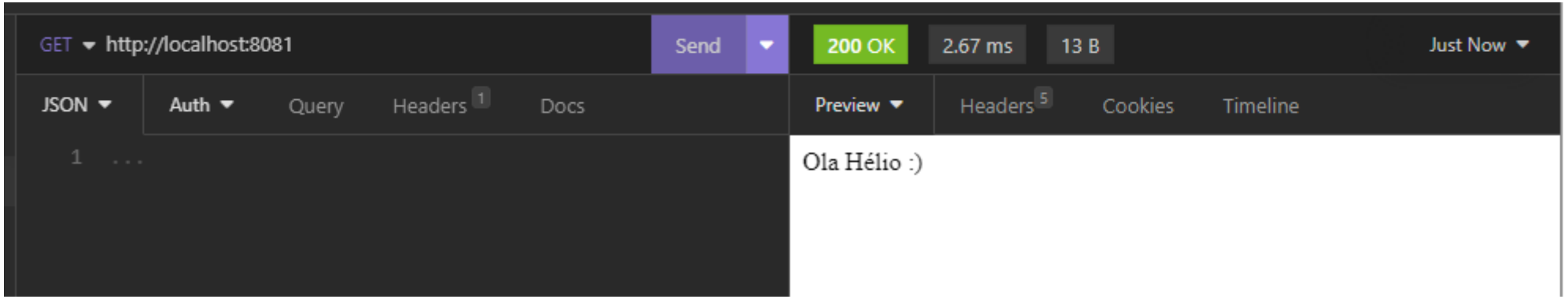
Crie o arquivo C:\apiTeste\app.py

```
from flask import Flask

app = Flask("Minha API")

#rota para a raiz: localhost:8081/
@app.route('/') #função que tratará a rota
def inicio():
    return 'Ola Hélio :)',200
```

Como rodar a aplicação?

- No console vá até a pasta onde está salvo o arquivo app.py (C:\apiTeste)
- Digite no console:
    - flask --app app.py --debug run -p 8081
- --app: defina o nome do arquivo que será executado.
- --debug: flag que define modo debug, utilizado para os desenvolvedores testem suas aplicações. Faz reload toda vez que acontecer uma modificação em seu código fonte.
- -p: Define a porta onde o serviço irá rodar.
    - Ps. Não rodar na porta 8080 nos laboratórios pois essa porta é para o apache.
    - Não rodar na porta 80 nos laboratórios.
    - Tipicamente o flask pode rodar na porta 5000/3000/8081.

Para testar utilize o insomnia



Exemplo: API Retângulo

Método	Rota	Corpo requisição	Retorno API

Get	/retangulos/areas	{
		"largura":5,
		"altura":10
		}
			{
			"area": 50
			}

	/retangulos/perimetros	{
		"largura":5,
		"altura":10
		}
			{
			"perimetro": 30
			}

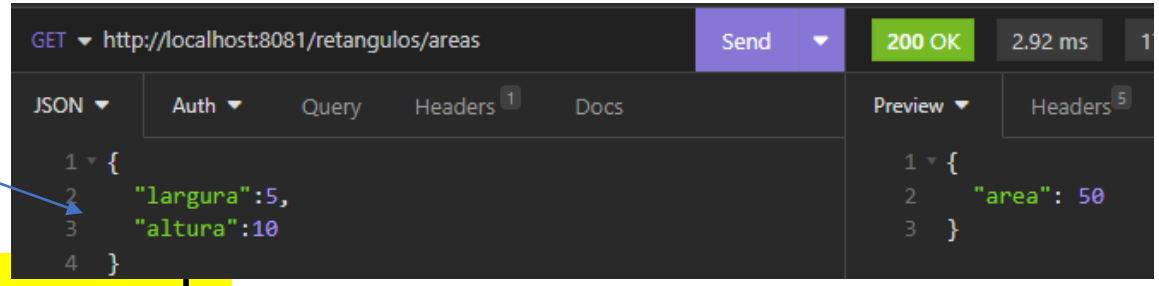
	/retangulos/diagonais	{
		"largura":5,
		"altura":10
		}
			{
			"diagonal": 11.180339887498949
			}

App.py – 1/4

```
from flask import Flask      #
from flask import request #para trabalhar com a requisição http
from flask import jsonify #para converter em json
import math


app = Flask("Minha API") #cria uma instância de flask
```

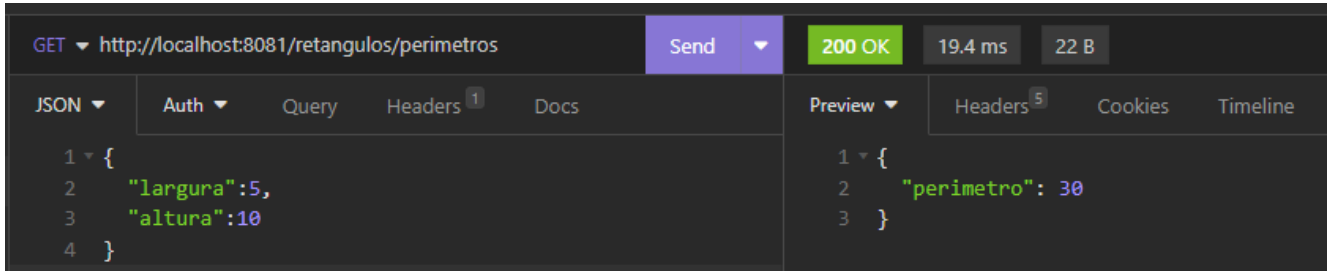
App.py – 2/4

```
@app.route('/retangulos/areas',methods=['GET']) #define a rota e o método http
def retangulos_areas(): #define a função que tratará a rota acima
    dados = request.json #recupera dados que vieram no corpo da requisição http
    largura = dados['largura']
    altura = dados['altura']
    caluloArea= largura*altura
    jsonResposta = jsonify( #monta json de resposta
        area=caluloArea
    )
    return jsonResposta ,200
```



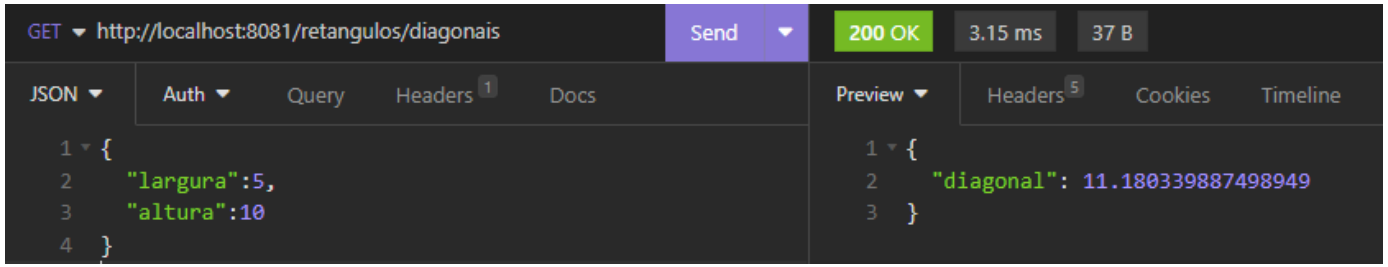
App.py – 3/4

```
@app.route('/retangulos/perimetros',methods=['GET'])
def retangulos_perimetros():
    dados = request.json
    largura = dados['largura']
    altura = dados['altura']
    caluloPerimetro= 2* largura+ 2*altura
    jsonResposta = jsonify(
        perimetro=caluloPerimetro
    )
    return jsonResposta ,200
```

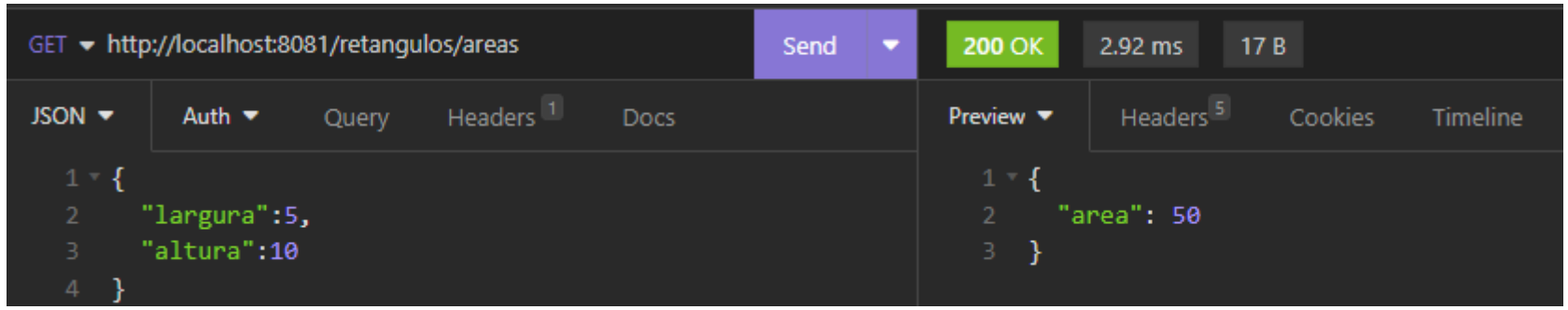


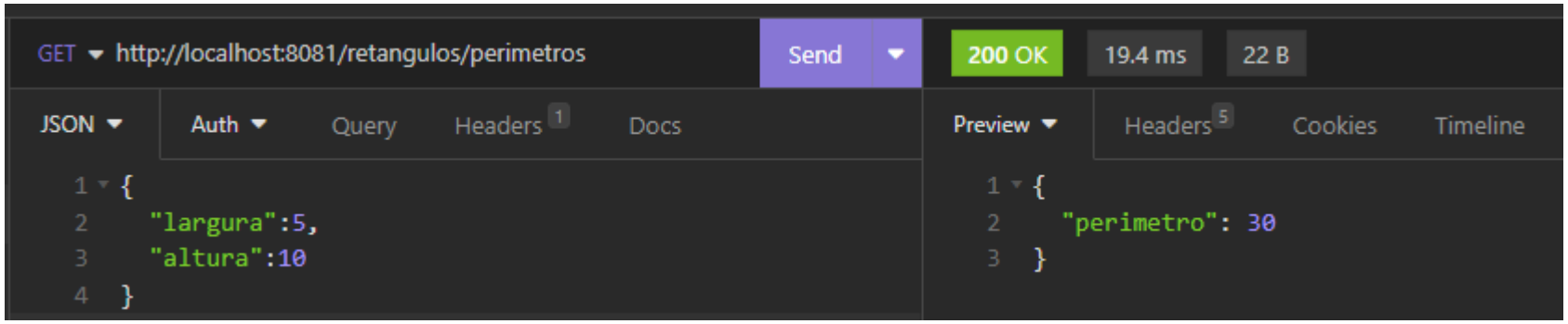
App.py – 4/4

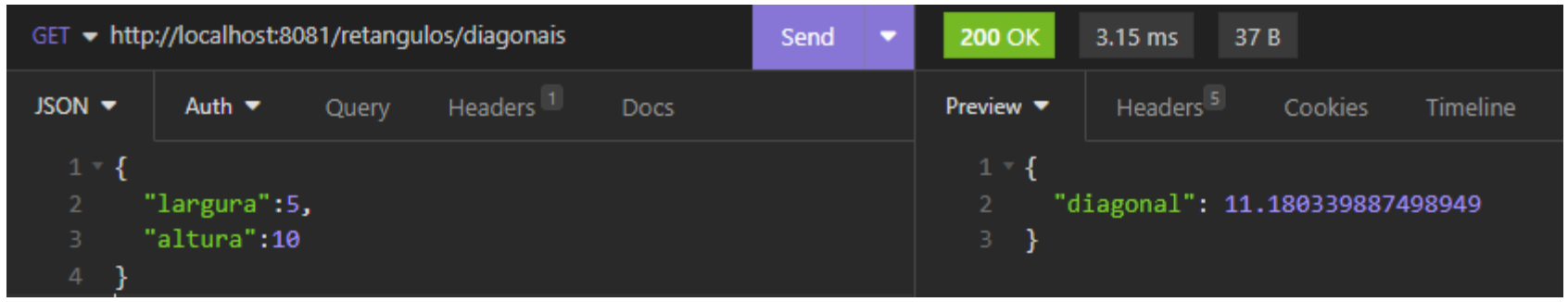
```
@app.route('/retangulos/diagonais',methods=['GET'])
def retangulos_diagonais():
    dados = request.json
    largura = dados['largura']
    altura = dados['altura']
    caluloDiagonal= math.sqrt(largura*largura + altura*altura)
    jsonResposta = jsonify(
        diagonal=caluloDiagonal
    )
    return jsonResposta ,200
```



testes


GET http://localhost:8081/retangulos/areas
Send
200 OK
2.92 ms
17 B
JSON
Auth
Query
Headers 1
Docs
Preview
Headers 5
Cookies
Timeline
{
"largura":5,
"altura":10
}
{
"area": 50
}


GET http://localhost:8081/retangulos/perimetros
Send
200 OK
19.4 ms
22 B
JSON
Auth
Query
Headers 1
Docs
Preview
Headers 5
Cookies
Timeline
{
"largura":5,
"altura":10
}
{
"perimetro": 30
}


GET http://localhost:8081/retangulos/diagonais
Send
200 OK
3.15 ms
37 B
JSON
Auth
Query
Headers 1
Docs
Preview
Headers 5
Cookies
Timeline
{
"largura":5,
"altura":10
}
{
"diagonal": 11.180339887498949
}

Parâmetros pela URI

Rota	Corpo requisição	Retorno API

/retangulos/areas/largura/altura	/retangulos/areas/5/10	{
		"area": 50
		}

/retangulos/perímetros/largura/altura	/retangulos/perímetros/5/10	{
		"perimetro": 30
		}

/retangulos/diagonais/largura/altura	/retangulos/diagonais/5/10	{
		"diagonal": 11.180339887498949
		}

App.py – 1/4

```
from flask import Flask
from flask import request
from flask import jsonify
import math
```

App.py – 2/4

```
@app.route('/retangulos/areas/<int:largura>/<int:altura>',methods=['GET'])
def retangulos_areas(largura,altura):
    caluloArea= largura*altura
    jsonResposta = jsonify(
        area=caluloArea
    )
    return jsonResposta ,200
```

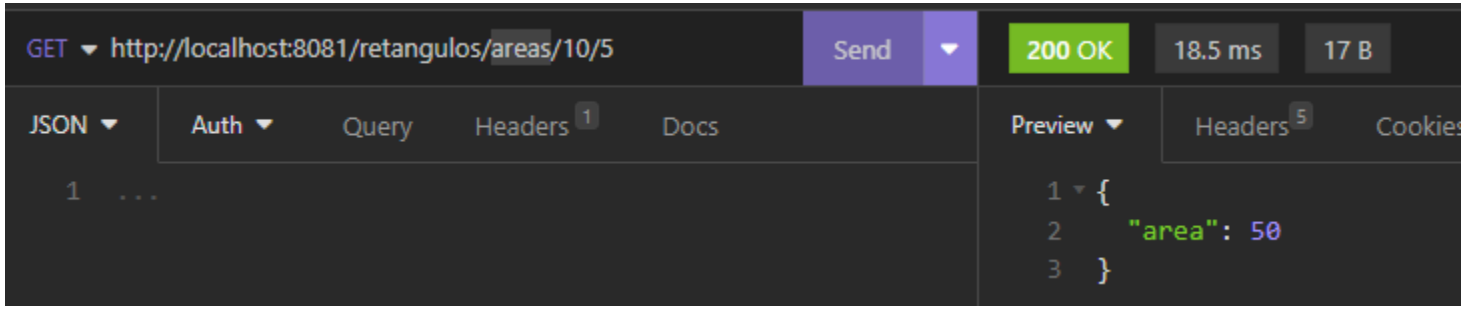
App.py – 3/4

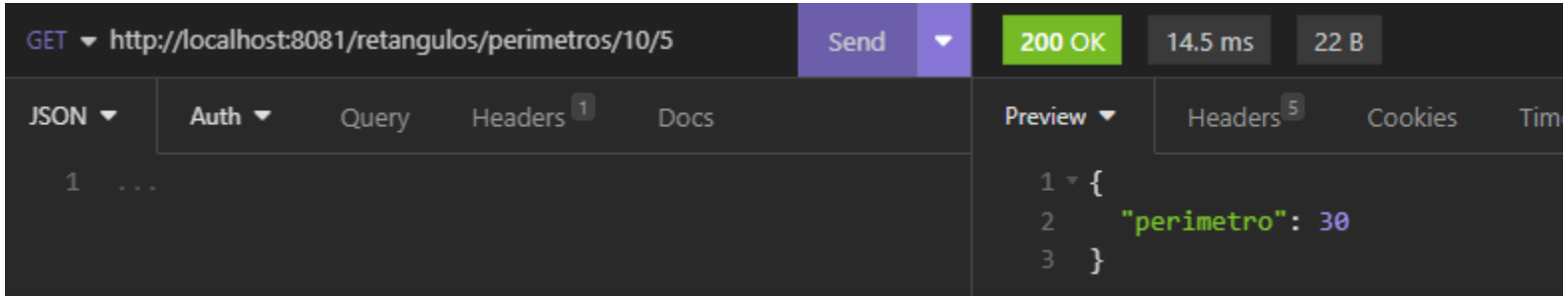
```
@app.route('/retangulos/perimetros/<int:largura>/<int:altura>',methods=['GET'])
def retangulos_perimetros(largura,altura):
    caluloPerimetro= 2* largura+ 2*altura
    jsonResposta = jsonify(
        perimetro=caluloPerimetro
    )
    return jsonResposta ,200
```

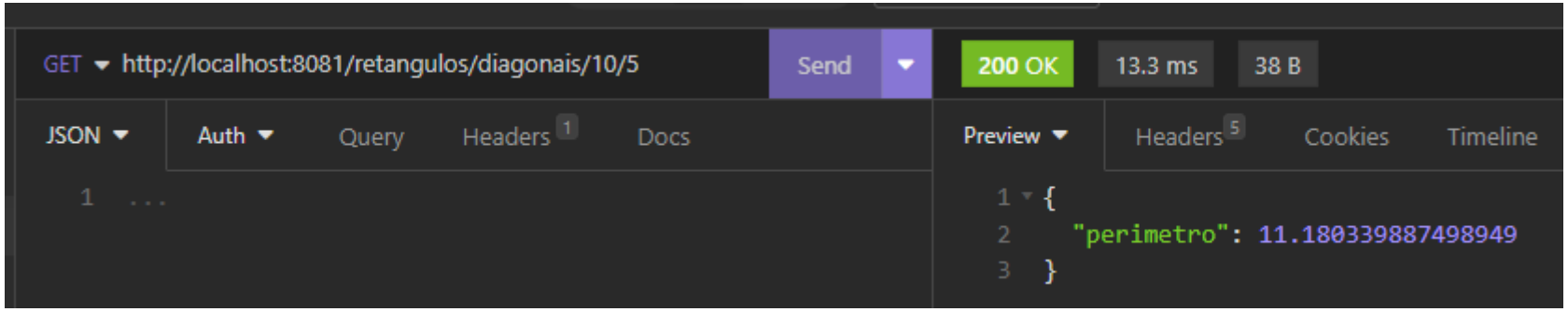
App.py – 4/4

```
@app.route('/retangulos/diagonais/<int:largura>/<int:altura>',methods=['GET'])
def retangulos_diagonais(largura,altura):

    caluloDiagonal= math.sqrt(largura*largura + altura*altura)
    jsonResposta = jsonify(
        diagonal=caluloDiagonal
    )
    return jsonResposta ,200
```

testes


GET http://localhost:8081/retangulos/areas/10/5
Send
200 OK
18.5 ms
17 B
JSON
Auth
Query
Headers 1
Docs
Preview
Headers 5
Cookies
1 ...
1 {
2 "area": 50
3 }


GET http://localhost:8081/retangulos/perimetros/10/5
Send
200 OK
14.5 ms
22 B
JSON
Auth
Query
Headers 1
Docs
Preview
Headers 5
Cookies
1 ...
1 {
2 "perimetro": 30
3 }


GET http://localhost:8081/retangulos/diagonais/10/5
Send
200 OK
13.3 ms
38 B
JSON
Auth
Query
Headers 1
Docs
Preview
Headers 5
Cookies
Timeline
1 ...
1 {
2 "perimetro": 11.180339887498949
3 }

API Banco de dados

- Instar o driver do mysql:
  - pip install mysql-connector-python

Tabela Cliente

```
CREATE SCHEMA IF NOT EXISTS aulaPHPmysql;
USE aulaPHPmysql;

CREATE TABLE IF NOT EXISTS aulaPHPmysql.Cliente (
  idCliente INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(128),
  email VARCHAR(45),
  senha VARCHAR(45),
  nascimento DATE,
  salario FLOAT
);
```

Rotas:

Método	Rota	Corpo requisição	Retorno API

post	/clientes	{
		"nome": "",
		"email":"",
		"senha":"",
		"salario": “”,
		"nascimento":""
		}
			Json do cliente cadastrado juntamente com seu id.

get	/clientes/		Array de clientes no formato json
get	/clientes/id	Id do cliente pesquisado	Json do cliente com id correspondente
Put	/clientes/id	Json com os dados que serão atualizados	
Delete	/clientes/id	Id do cliente que será excluído	

Arquivos:

- app.py
  - Inicializa o aplicativo
- rotasCliente.py
  - Trata as rotas referentes ao cliente.

App.py

```
from flask import Flask
from rotasCliente import rotasCliente
from flask import request
from flask import jsonify


app = Flask("Minha API")

# as rotas do cliente estão em outro arquivo(rotasCliente.py)
app.register_blueprint(rotasCliente) #registra as rotas do cliente.
```

rotasCliente.py

```
from flask import Blueprint
from mysql import connector
from flask import jsonify
from flask import json
from flask import request
#configura o "alfabeto" do json para trabalho com UTF8
json.provider.DefaultJSONProvider.ensure_ascii = False

#configura objeto para tratar as rotas do cliente.
rotasCliente = Blueprint('API', "Cliente")
```

rotasCliente.py

```
#cria uma conexao com o banco de dados
conexao = connector.connect(
  host="localhost",        #endereço do banco
  user="root",             #usuario
  password="",             #senha
  database="aulaPHPmysql",  #banco para conectar
  charset='utf8'          #'alfabeto'
)
```

rotasCliente.py

```
@rotasCliente.route('/clientes',methods=['POST'])
def clientes_POST():
    dados = request.json
    cursor = conexao.cursor(prepared=True)
    sql = """ insert into Cliente (nome, email,senha,nascimento,salario)
            values (%s,%s,%s,%s,%s); """
    novaTupla = (dados['nome'], dados['email'], dados['senha'], dados['nascimento'],dados['salario'])
    cursor.execute(sql, novaTupla)
    conexao.commit()
    idNovo = cursor.lastrowid
    cursor.close()
    jsonResposta = jsonify(
        idCliente=idNovo,
        nome = dados['nome'],
        email= dados['email'],
        nascimento= dados['nascimento'],
        salario = dados['salario']
    )
    return jsonResposta
```

rotasCliente.py

```
@rotasCliente.route('/clientes',methods=['GET'])
def clientes_GET():
    cursor = conexao.cursor()
    cursor.execute("""SELECT idCliente, nome, email,
DATE_FORMAT(nascimento,'%d/%m/%Y'),   salario from Cliente order by nome""");
    result = cursor.fetchall()
    cursor.close()
    conexao.commit()
    return jsonify(result),200
```

rotasCliente.py

```
@rotasCliente.route('/clientes/<int:idCliente>',methods=['GET'])# http://localhost:8081/1
def clientes_GET_idCliente(idCliente):
  cursor = conexao.cursor(prepared=True)
    sql = """SELECT idCliente,
                nome,
                email,
                DATE_FORMAT(nascimento,'%d/%m/%Y'),
                salario
                from Cliente where idCliente=%s order by nome"""
    selectTupla = (idCliente,)
    cursor.execute(sql, selectTupla)
    result = cursor.fetchall()
    cursor.close()
    conexao.commit()
    jsonResposta = jsonify(
       result
    )
    return jsonResposta
```

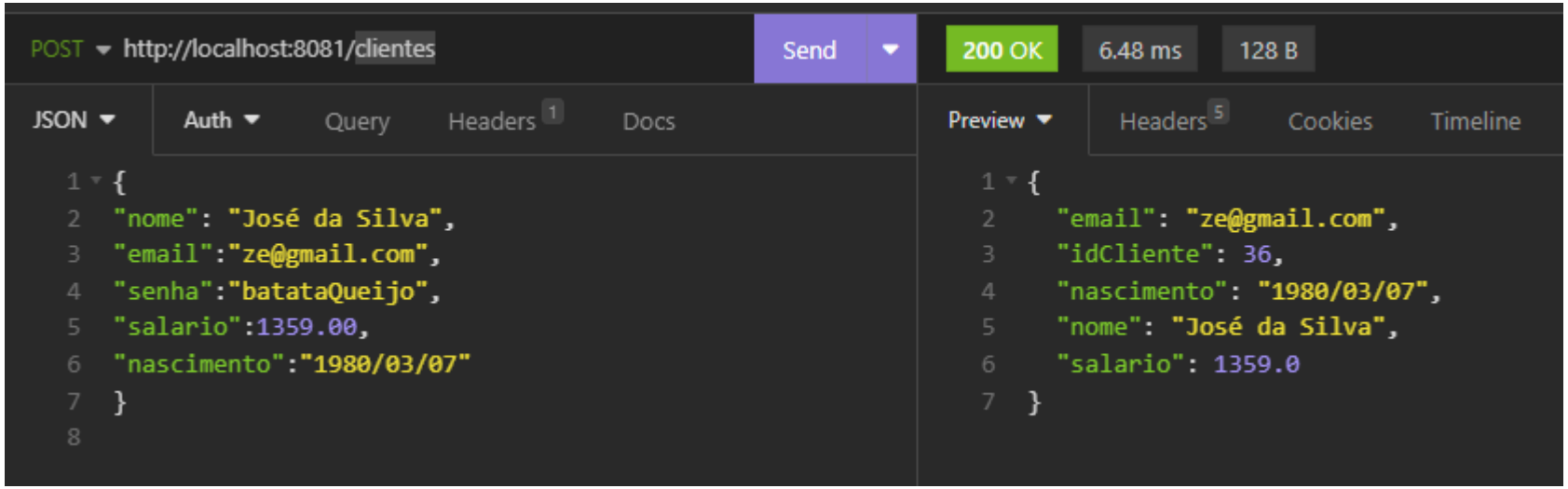
rotasCliente.py

```
@rotasCliente.route('/clientes/<int:idCliente>',methods=['PUT'])
def clientes_PUT(idCliente):
    dados = request.json
    cursor = conexao.cursor(prepared=True)
    sql = """ update cliente set nome=%s, email=%s, senha=%s, salario=%s, nascimento=%s
              where idCliente = %s """
    atualizarTupla = (dados['nome'], dados['email'], dados['senha'], dados['salario'],dados['nascimento'],
idCliente)
    cursor.execute(sql, atualizarTupla)
    conexao.commit()
    cursor.close()
    jsonResposta = jsonify(
        idCliente=idCliente,
        nome = dados['nome'],
        email= dados['email'],
        nascimento= dados['nascimento'],
        salario = dados['salario']
    )
    return jsonResposta
```

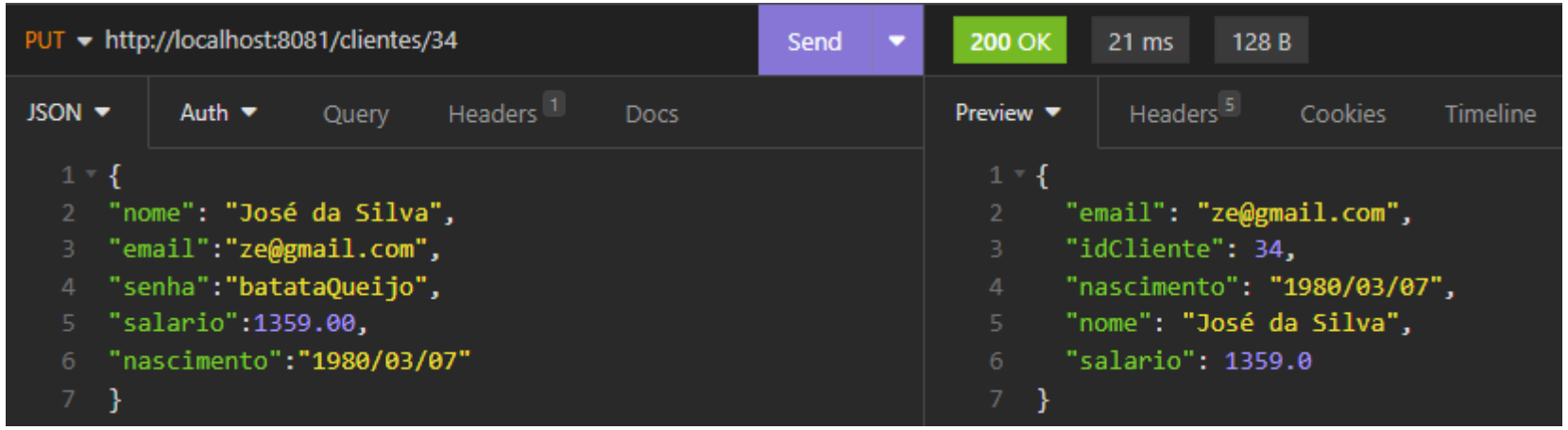
rotasCliente.py

```
@rotasCliente.route('/clientes/<int:idCliente>',methods=['DELETE'])
def clientes_DELETE(idCliente):
    cursor = conexao.cursor(prepared=True)
    sql = "delete from cliente where idcliente=%s"
    excluirTupla = (idCliente,)
    cursor.execute(sql, excluirTupla)
    conexao.commit()
    cursor.close()

    jsonResposta = jsonify(
        status='ok',
        msg='Excluído com sucesso!'

    )
    return jsonResposta
```

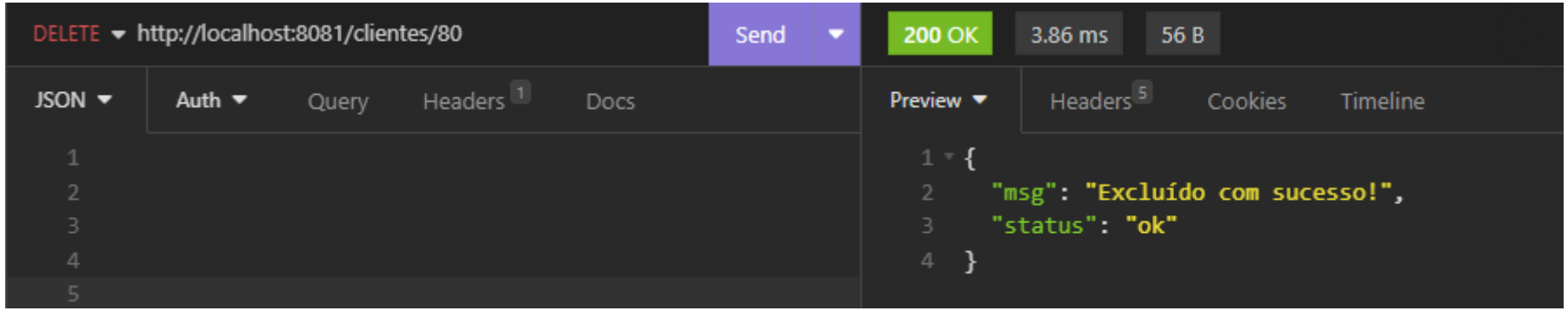
Teste
post: /clientes



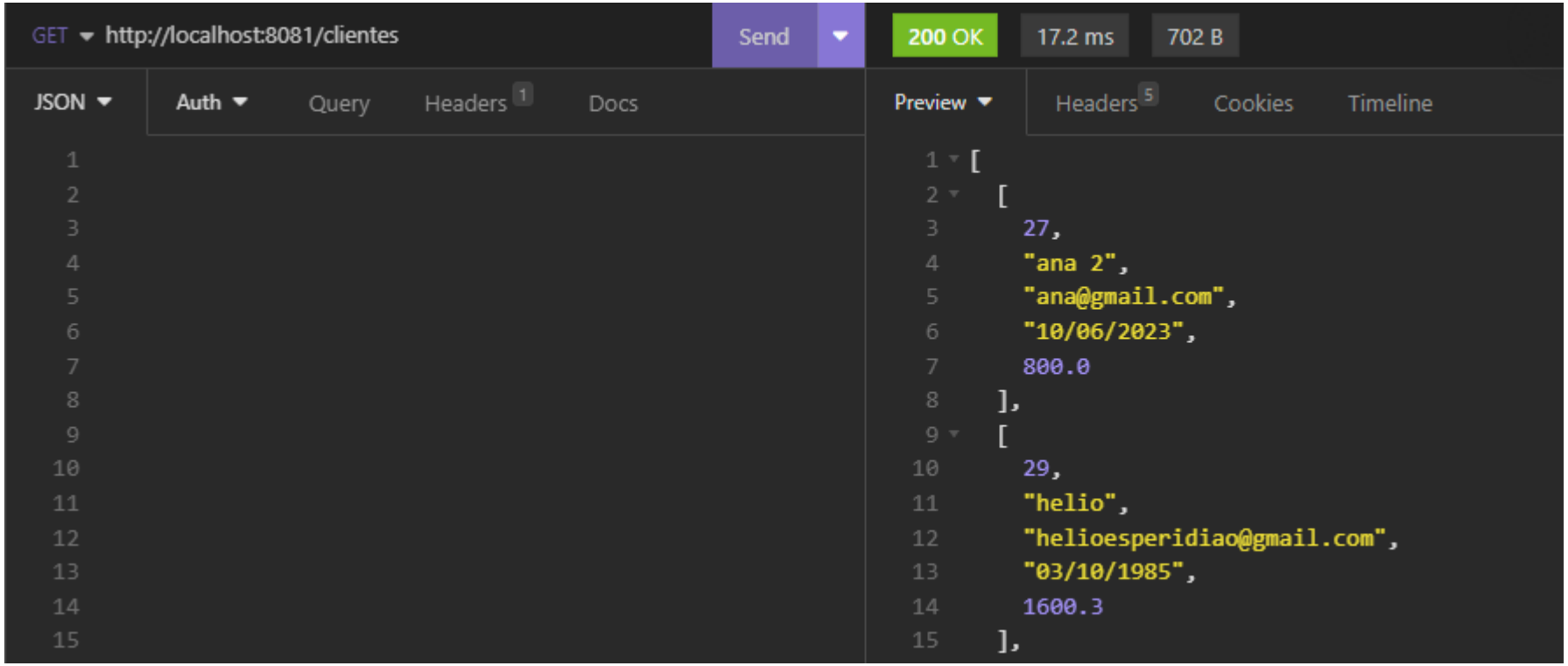
Teste
put: /clientes



Teste
delete: /clientes/idCliente



Teste
get: /clientes



Teste
get: /clientes/idCliente